

```

Procedure INIT-PATH
{Invoked when the node comes up.}
1. Initialize all tables.
2. Run PATH algorithm.
End INIT-PATH

Algorithm PATH
{Invoked when a message M is received from neighbor k,
or an adjacent link to k has changed or when a node is
initialized.}
1. Run NTU to update neighbor tables.
2. Run MTU to update main tables.
3. For each destination j marked as changed,
    Add update entry  $[j, D_p^i, p_j^i]$  to the new message  $M'$ .
4. Within finite amount of time, send message  $M'$  to
    each neighbor.
End PATH

```

FIG. 1

```

Procedure NTU
{Called by PATH to process an event.}
1. If event is a message M from neighbor k,
    a. For each entry  $[j, d, p]$  in M //Note  $d = D_p^k, p = p_j^k$ )
        Set  $D_{jk}^i \leftarrow d$  and  $p_{jk}^i \leftarrow p$ 
    b. For each destination j with an entry in M,
        Remove existing links  $(n, j)$  in  $T_k^i$  and add new
        link  $(m, j, d)$  to  $T_k^i$ , where  $d = D_{jk}^i - D_{mk}^i$ 
        and  $m = p_{jk}^i$ .
2. If the event is an adjacent link-status change, update  $l_k^i$  and
    clear neighbor tables of k, if link is down.
End NTU

```

FIG. 2

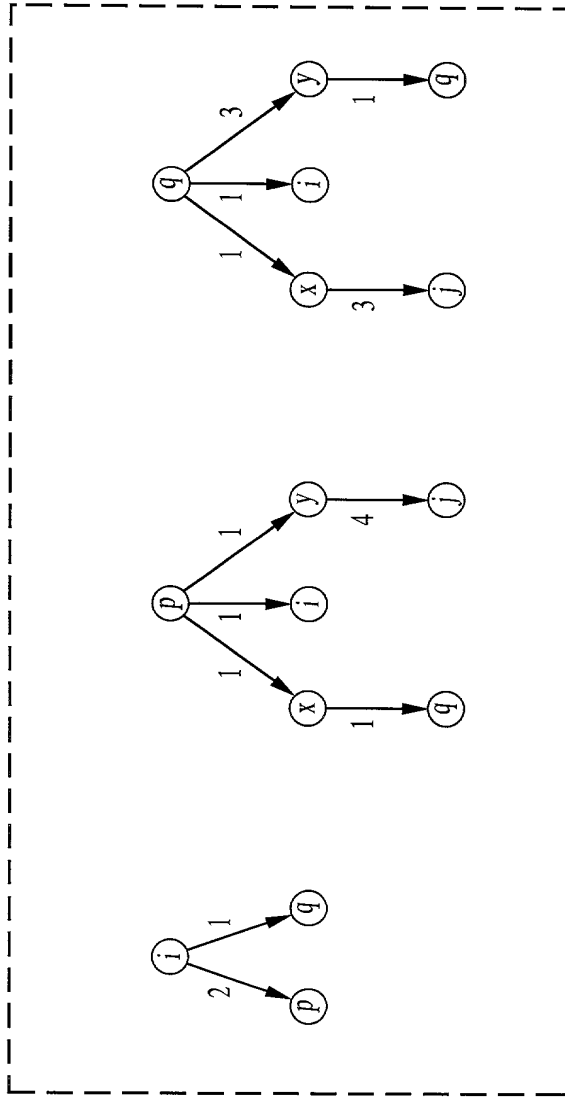
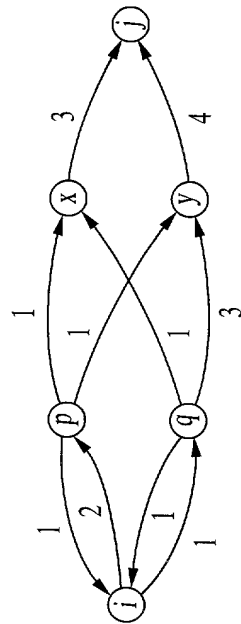


FIG. 3A



Preferred Neighbors

Destination	p	q	x	y	j
Distance	2	1	2	3	5
Pref. Nbr	p	q	q	p	q

FIG. 3B

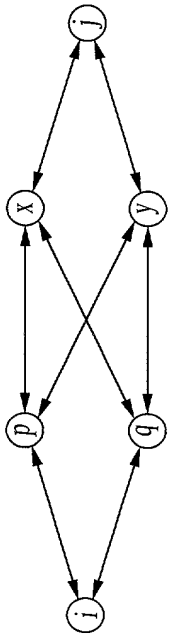


FIG. 4A

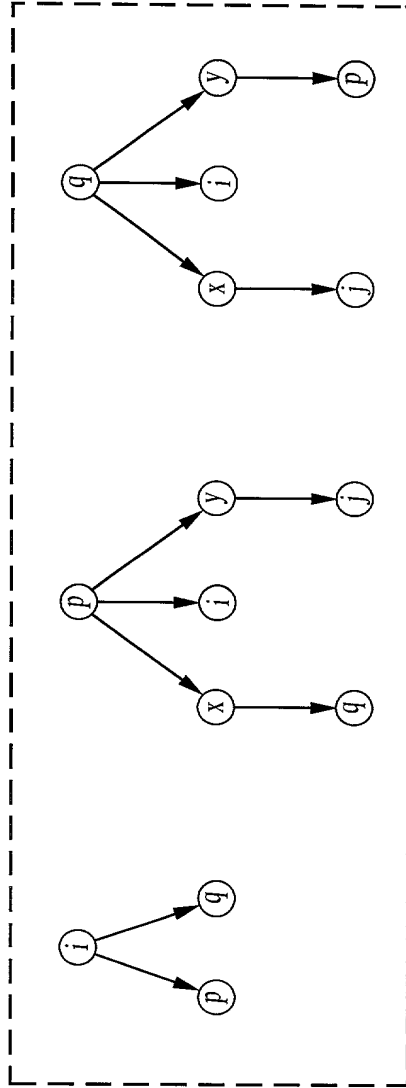


FIG. 4B

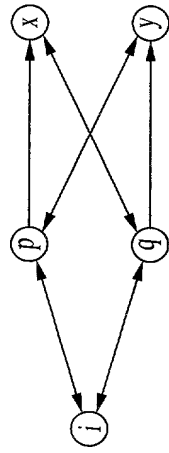


FIG. 4C

Procedure *MTU*

1. Clear link table T^i
2. For each node $j \neq i$ occurring in at least one T_k^i
 - a. Find $MIN \leftarrow \min\{D_{jk}^i + l_k^i | k \in N^i\}$.
 - b. Let n be such that $MIN = (D_{jn}^i + l_n^i)$. Ties are broken *consistently*. Neighbor n is the preferred neighbor for destination j . For each link (j, v, d) in T_n^i Add link (j, v, d) to T^i
3. Update T with each link l_k^i
4. Run Dijkstra's shortest path algorithm on T^i to find new D_j^i , and p_j^i
5. For each destination j , if D_j^i or p_j^i changed from previous value, set *changed* and *report-it* flags for j .

End *MTU*

FIG. 5

Procedure *INIT-MPATH*

{*Invoked when the node comes up.*}

1. Initialize tables and run *MPATH*.

End *INIT-MPATH*

Algorithm *MPATH*

{*Invoked when a message M is received from neighbor k , or an adjacent link to k has changed.*}

1. Run *NTU* to update neighbor tables.
2. Run *MTU* to obtain new D_j^i and p_j^i .
3. If node is *PASSIVE* or node is *ACTIVE* \wedge last reply arrived,

Reset *goactive* flag.

For each destination j marked as *report-it*,

- a. $FD_j^i \leftarrow \min\{D_j^i, RD_j^i\}$
- b. If $D_j^i > RD_j^i$, Set *goactive* flag.
- c. $RD_j^i \leftarrow D_j^i$
- d. Add $[j, RD_j^i, p_j^i]$ to message M' .
- e. Clear *report-it* flag for j .

Otherwise, the node is *ACTIVE* and waiting for more replies,

For each destination j marked as *changed*,

- f. $FD_j^i \leftarrow \min\{D_j^i, FD_j^i\}$

4. For each destination j marked as *changed*,

a. Clear *changed* flag for j

b. $S_j^i \leftarrow \{k | D_{jk}^i < FD_j^i\}$

5. For each neighbor k ,

a. $M'' \leftarrow M'$

b. If event is *query* from k , Set *reply* flag in M'' .

c. If *goactive* set, Set *query* flag in M'' .

d. If M'' non-empty, send M'' to k .

6. If *goactive* set, become *ACTIVE*, otherwise become *PASSIVE*.

END *MPATH*

FIG. 6